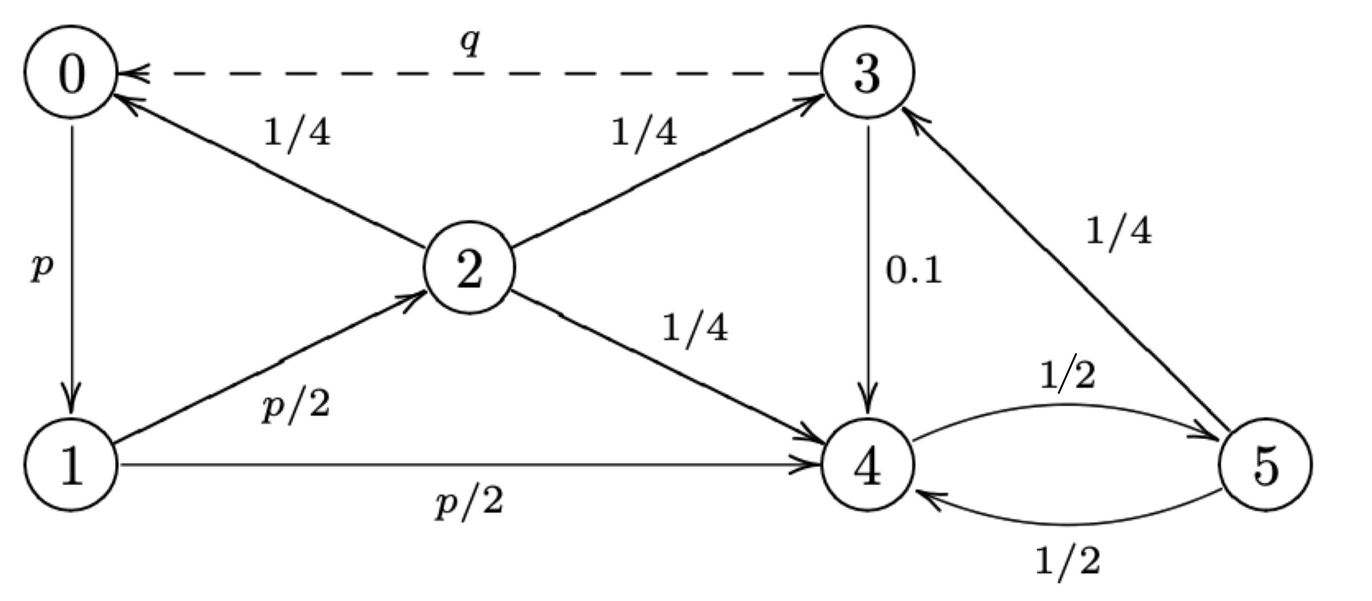
Stochastic Systems

Markov Chains

Gloria del Valle Cano

In [1]:
```python
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from IPython.display import display, Math, Latex
```

Consider the following Markov Chain:



It is assumed that in each state there is an arrow to the same state that makes the sum of the output probabilities 1.

i) Determine the transition matrix of this chain.

In [2]:
```python
def t_matrix(p, q):
    """
    Returns transition probabilities for the markov model given p and q.
    Args:
        p (float): basic parameter for the transition.
        q (float): probability based on these values.
    """
                #      0      1       2       3       4       5
    return np.array([[  1-p,    p,    0.0,    0.0,    0.0,    0.0], # 0
                     [  0.0,  1-p,  p/2.0,    0.0,   p/2.0,   0.0], # 1
                     [1/4.0,  0.0,  1/4.0,   1/4.0,  1/4.0,   0.0], # 2
                     [    q,  0.0,    0.0, 1-q-0.1,    0.1,   0.0], # 3
                     [  0.0,  0.0,    0.0,     0.0,  1/2.0, 1/2.0], # 4
                     [  0.0,  0.0,    0.0,   1/4.0, 1/2.0, 1/4.0]  # 5
                     ])
```

ii) Simulate the operation of the chain and make an overall estimation of h_o^2 and h_0^5 for $q = 0.1$ and $q = 0$.

iii) Simulate the operation of the chain and make an overall estimation of k_o^2 and k_4^2 for $q = 0.1$ and $q = 0$.

Let's create a Markov class that computes the probabilities given by P and simulates every step. This class also returns the hitting probability and the hitting time for reaching whichever states from a determined initial state, simulating nc independent chains until ns steps. Thus, it is obtained the mean hitting time, coded according to the following principles.

Let $(X_t)_{t\geq 0}$ be a Markov chain with transition matrix P. The hitting time of a subset A is the random variable: $H^A : \Omega \to \{0, 1, 2, \dots\} \cup \{\infty\}$ given by

$$H^A(\omega) = \inf\{t \geq 0 : X_n(\omega) \in A\}$$

where we agree that the infimum of the empty set 0 is ∞. The probability starting from i that $(X_t)_{t\geq 0}$ ever hits A is then

$$h_i^A = P_i(H^A < \infty).$$

When A is closed class, h_i^A is called the *absorption probability*. The mean time taken for $(X_t)_{t\geq 0}$ to reach A is given by

$$k_i^A = E_i(H^A) = \sum_{t<\infty} tP(H^A = t) + \infty P(H^A = \infty)$$

In [3]:
```python
class Markov:
    """
    Class that computes a Markov model using the transition
    matrix P starting from a specific state.
    """
    def __init__(self, P, m0):
        """
        Init function for Markov model.
        Args:
            P (np array): transition matrix.
            m0 (int): initial state.
        """
        self.P = P                # P
        self.nst = len(self.P[0])  # number of states
        self.st = m0               # initial state

    def transition(self, st):
        """
        Determines a new state with the corresponding
        probabilities returning the value of the new state,
        generating a random event sampled from the uniform
        distribution.
        Args:
            st (int): actual state.
        """
        return np.where(np.random.uniform() < np.cumsum(self.P[st]))[0][0]

    def history(self, nst, m):
        """
        Returns the probability occupation of the previous state.
        Args:
            nst (int): number of states.
            m (int): probability of previous state.
        """
        return [m for i in range(nst)]

    def h__(self, nc, ns):
        """
        Computes the hitting probability to reach
        whichever states from initial state.
        Args:
            nc (int): number of chains.
            ns (int): number of states.
        """
        sts = np.zeros((nc, ns))
        sts[:, self.st] = 1
        hist = self.history(nc, self.st)
        for s in range(ns):
            for c in range(nc):
                st = self.transition(hist[c])
                sts[c, st] = 1
                hist[c] = st
        return sts

    def k__(self, nc, ns):
        """
        Computes the hitting time to reach
        whichever states from initial state.
        Args:
            nc (int): number of chains.
            ns (int): number of states.
        """
        sts = np.full((nc, ns), np.inf)
        sts[:, self.st] = 0
        hist = self.history(nc, self.st)
        for s in range(ns):
            for c in range(nc):
                st = self.transition(hist[c])
                if sts[c, st] == np.inf:
                    sts[c, st] = s
                hist[c] = st
        return sts
```

- P for $q = 0.1$

In [4]:
```python
p = 0.3
q = 0.1
P = t_matrix(p, q)
```

We compute the hitting probabilities for $q = 0.1$, and, for instance $nc = 2000$ (number of chains) and $ns = 2000$ (number of steps).

In [5]:
```python
states_h0 = Markov(P, 0).h__(2000, 2000)
states_k0 = Markov(P, 0).k__(2000, 2000)
states_k4 = Markov(P, 4).k__(2000, 2000)
```

Then, we obtain the average of the hitting probabilities and the mean hitting time for each case (h_0^2, h_0^5, k_0^2 and k_4^2).

In [6]:
```python
display(Latex(r'Hitting probabilities and hitting mean time for $q = {0}$'.format(q)))
display(Math(r'h_0^2 = {0}'.format(states_h0.mean(axis=0)[2])))
display(Math(r'h_0^5 = {0}'.format(states_h0.mean(axis=0)[5])))
display(Math(r'k_0^2 = {0}'.format(states_k0.mean(axis=0)[2])))
display(Math(r'k_4^2 = {0}'.format(states_k4.mean(axis=0)[2])))
```

Hitting probabilities and hitting mean time for $q = 0.1$

$h_0^2 = 1.0$

$h_0^5 = 1.0$

$k_0^2 = 44.6825$

$k_4^2 = 74.019$

- P for $q = 0.0$

In [7]:
```python
p = 0.3
q = 0.0
P = t_matrix(p, q)
```

As before, let's compute the hitting probabilities for $q = 0.0$, and again $nc = 2000$ (number of chains) and $ns = 2000$ (number of steps).

In [8]:
```python
states_h0 = Markov(P, 0).h__(2000, 2000)
states_k0 = Markov(P, 0).k__(2000, 2000)
states_k4 = Markov(P, 4).k__(2000, 2000)
```

Finally, we obtain the average of the hitting probabilities and the mean hitting time for each case (h_0^2, h_0^5, k_0^2 and k_4^2).

In [9]:
```python
display(Latex(r'Hitting probabilities and hitting mean time for $q = {0}$'.format(q)))
display(Math(r'h_0^2 = {0}'.format(states_h0.mean(axis=0)[2])))
display(Math(r'h_0^5 = {0}'.format(states_h0.mean(axis=0)[5])))
display(Math(r'k_0^2 = {0}'.format(states_k0.mean(axis=0)[2])))
display(Math(r'k_4^2 = {0}'.format(states_k4.mean(axis=0)[2])))
```

Hitting probabilities and hitting mean time for $q = 0.0$

$h_0^2 = 0.501$

$h_0^5 = 1.0$

$k_0^2 = inf$

$k_4^2 = inf$

iv) Use the appropriate system of linear equations to determine the theoretical values corresponding to the estimated quantities and compare with the values determined by simulation (caution: if a quantity k is ∞ the simulation clearly cannot give its real value... discuss this case).

- $q = 0.1$.
  - h_0^2:

$$\begin{cases} h_0 = (1-p)h_0 + ph_1 \\ h_1 = (1-p)h_1 + \frac{p}{2}h_2 + \frac{p}{2}h_4 \\ h_2 = 1 \\ h_3 = qh_0 + (0.9-q)h_3 + 0.1h_4 \\ h_4 = \frac{1}{2}h_4 + \frac{1}{2}h_5 \\ h_5 = \frac{1}{4}h_3 + \frac{1}{2}h_4 + \frac{1}{4}h_5 \end{cases} \longrightarrow h_0^2 = 1$$

  - h_0^5:

$$\begin{cases} h_0 = (1-p)h_0 + ph_1 \\ h_1 = (1-p)h_1 + \frac{p}{2}h_2 + \frac{p}{2}h_4 \\ h_2 = \frac{1}{4}h_0 + \frac{1}{4}h_2 + \frac{1}{4}h_3 + \frac{1}{4}h_4 \\ h_3 = qh_0 + (0.9-q)h_3 + 0.1h_4 \\ h_4 = \frac{1}{2}h_4 + \frac{1}{2}h_5 \\ h_5 = 1 \end{cases} \longrightarrow h_0^5 = 1$$

  - k_0^2 and k_4^2 (*):

$$\begin{cases} k_0 = 1 + (1-p)k_0 + pk_1 \\ k_1 = 1 + (1-p)k_1 + \frac{1}{2}k_2 + \frac{1}{2}k_4 \\ k_2 = 0 \\ k_3 = 1 + qk_0 + (0.9-q)k_3 + 0.1k_4 \\ k_4 = 1 + \frac{1}{2}k_4 + \frac{1}{2}k_5 \\ k_5 = 1 + \frac{1}{4}k_3 + \frac{1}{2}k_4 + \frac{1}{4}k_5 \end{cases} \longrightarrow \begin{cases} k_0^2 = 43.33 \\ k_4^2 = 73.33 \end{cases}$$

(*) Last case can be easily resolved with *numpy*, because it's a determined system with unique solution:

In [10]:
```python
q = 0.1
 #       0       1       2        3        4        5
M_2 = [[1-p-1,     p,    0.0,    0.0,    0.0,       0.0], # 0
      [  0.0,  1-p-1,  p/2.0,    0.0,   p/2.0,      0.0], # 1
      [  0.0,    0.0,      1,    0.0,     0.0,      0.0], # 2
      [    q,    0.0,    0.0, 0.9-q-1,    0.1,      0.0], # 3
      [  0.0,    0.0,    0.0,    0.0,   1/2.0-1,  1/2.0], # 4
      [  0.0,    0.0,    0.0,   1/4.0,  1/2.0,  1/4.0-1]] # 5

display(Latex(r"For $q=0.1$:"))
display(Math(r"k_0^2 = {0}".format(np.linalg.solve(M_2, -np.array([1,1,0,1,1,1]))[0])))
display(Math(r"k_4^2 = {0}".format(np.linalg.solve(M_2, -np.array([1,1,0,1,1,1]))[4])))
```

For $q = 0.1$:

$k_0^2 = 43.333333333333336$

$k_4^2 = 73.33333333333337$

These results match the simulation values.

- $q = 0$. (So $\{3, 4, 5\}$ is a closed class).
  - h_0^2:

$$\begin{cases} h_0 = (1-p)h_0 + ph_1 \\ h_1 = (1-p)h_1 + \frac{p}{2}h_2 \\ h_2 = 1 \\ h_3 = h_4 = h_5 = 0 \end{cases} \longrightarrow h_0^2 = \frac{1}{2}$$

  - h_0^5:

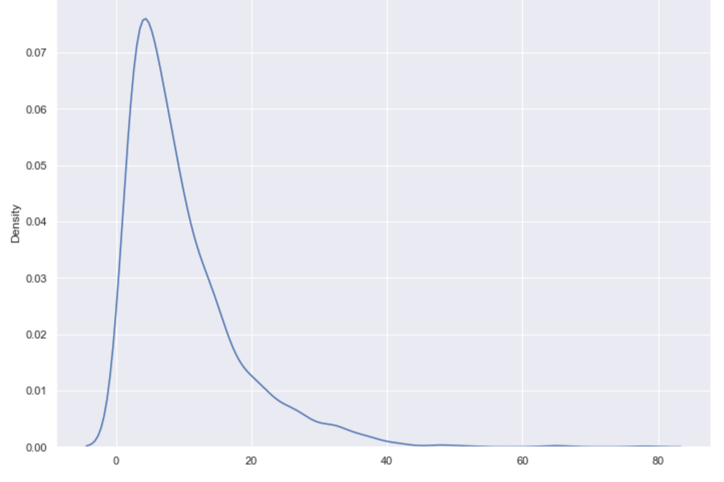
$$\begin{cases} h_0 = (1-p)h_0 + ph_1 \\ h_1 = (1-p)h_1 + \frac{p}{2}h_2 + \frac{p}{2}h_4 \\ h_2 = \frac{1}{4}h_0 + \frac{1}{4}h_2 + \frac{1}{4}h_3 + \frac{1}{4}h_4 \\ h_3 = 0.9h_3 + 0.1h_4 \\ h_4 = \frac{1}{4}h_4 + \frac{1}{2}h_5 \\ h_5 = 1 \end{cases} \longrightarrow h_0^5 = 1$$

  - k_0^2 and k_4^2 (indeterminate system):

$$\begin{cases} k_0 = 1 + (1-p)h_0 + ph_1 \\ k_1 = 1 + (1-p)h_1 + \frac{p}{2}h_2 + \frac{p}{2}h_4 \\ k_2 = 0 \\ k_3 = k_4 = k_5 = \infty \end{cases} \longrightarrow \begin{cases} k_0^2 = \infty \\ k_4^2 = \infty \end{cases}$$

v) For $q = 0.1$, plot $g(t) = P[H_0^{\{4\}} = t]$

In [11]:
```python
# Initial values
p = 0.3
q = 0.1
P = t_matrix(p, q)
# Create Markov chain from the state 0 and return probabilities
states_H0 = Markov(P, 0).k__(2000, 2000)
# Plot kde (density estimation) for k probability for state 4
sns.set(rc={'figure.figsize':(11.7, 8.27)})
sns.kdeplot(states_H0[:,4])
plt.show()
```



We see that in $t = 0$ we are in state 0 so our probability is 0. When we get the state 4 for first time the probability increases until it reaches the maximum and then our probability to reach again the state 4 decreases.

vi) Affirming $H_0^{\{4\}} < H_0^{\{5\}}$. Is it true/false? Why? (Choose the appropriate option).

It's true because it's necessary to get first the state 4 to get the state 5. Therefore, the time it takes to reach state 5 from state 4 will always be at least 1 time unit after. This is corroborated by:

In [12]:
```python
(states_H0[:,4] < states_H0[:,5]).all()
```

Out[12]:
```
True
```

In [13]:
```python
from IPython.core.display import HTML
HTML("""
<style>
    .qst {
        background-color: #E2EAF5;
        padding:25px;
        border-radius: 5px;
        border: solid 2px #5D8AA8;
    }

    .qst:before {
        font-weight: bold;
        display: block;
        margin: 0px 10px 10px 10px;
    }

    .qst2 {
        background-color: #F2ECD9;
        padding:25px;
        border-radius: 5px;
        border: solid 2px #E7CE78;
    }

    .qst2:before {
        font-weight: bold;
        display: block;
        margin: 5px 10px 10px 10px;
    }
    </style>
""")
```

Out[13]: